RELATÓRIO ESINF - SPRINT 2

DIAGRAMA DE CLASSES E COMPLEXIDADE DE ALGORITMOS

GRUPO 21 – 2DC

1191507 – Bárbara Pinto

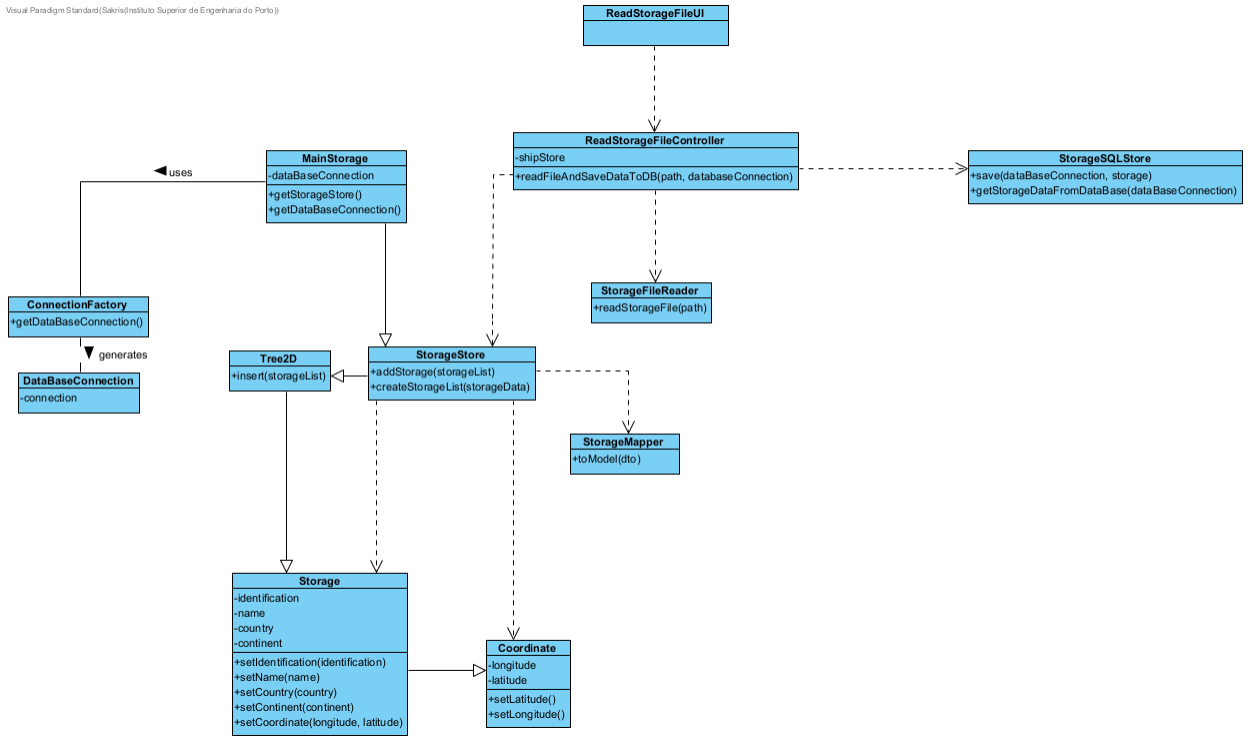
1200991 - Carlos Dias

1201029 - Cristóvão Sampaio

1201045 - Miguel Silva

1201154 – Martim Maciel

US201



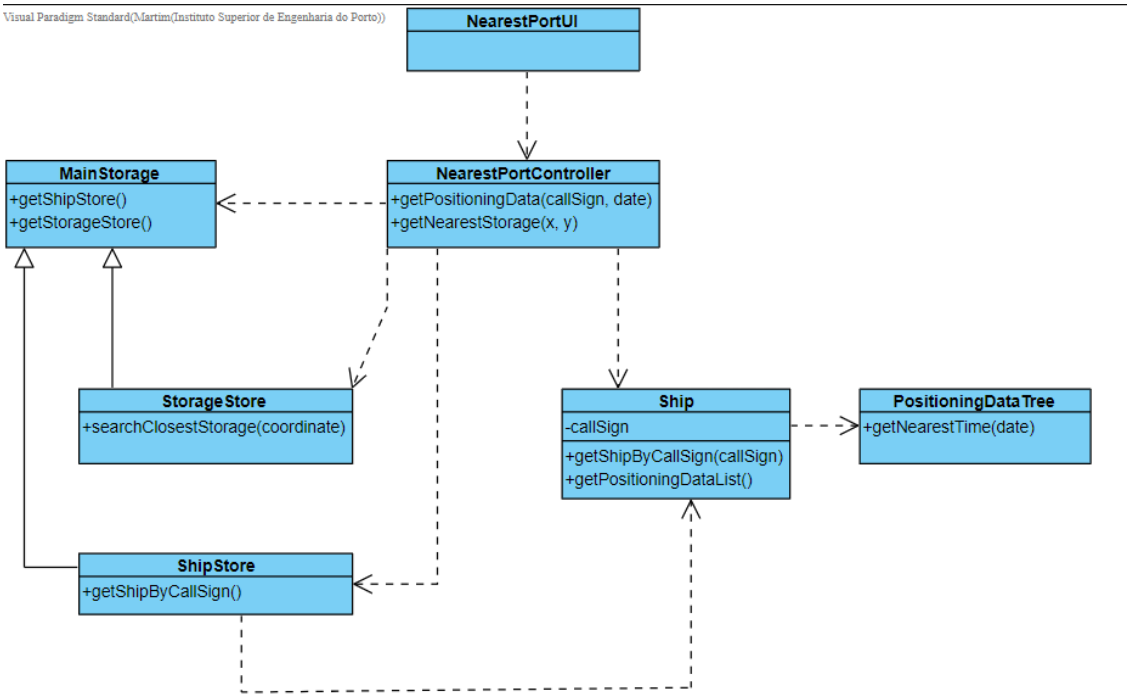
A User Story 201 pede para importar um ficheiro com dados de portos e inserir esses mesmos dados numa 2D-Tree balanceada.

A primeira coisa que o algoritmo vai ter de fazer é iterar pelo ficheiro e retirar todos os portos que se encontram lá, criando os objetos a partir desses dados e guardando esses mesmos objetos numa lista $\boldsymbol{O(1)}$. Esta primeira parte terá uma complexidade de $\boldsymbol{O(n)}$ visto que vai ser necessário passar por todos os ports que se encontram dentro do ficheiro.

Após ter os dados guardados numa lista, a próxima coisa que vai ser necessário fazer é inserir esses dados na 2D-Tree, aqui como estamos a trabalhar com uma 2D-Tree balanceada, há uns cuidados extra que temos de ter. Primeiramente vamos trabalhar com uma lista de dados, então temos de encontrar a mediana desta lista, pois aí será a root da respetiva subárvore. Para encontrar a mediana, primeiro é necessário dar sort à lista, para isso usamos o *Collections.sort()* que tem uma complexidade de $\boldsymbol{O(nlog(n))}$ no pior caso, após isso podemos simplesmente usar o método *size()* que tem de complexidade $\boldsymbol{O(1)}$ e fazer uma conta aritmética para encontrar qual o índice do elemento mediano da lista sendo esse elemento guardado num node, após isso será necessário criar 2 sublistas, uma apenas com os elementos à esquerda da mediana, e outra apenas com os à direita, e fazemos uma chamada recursiva ao algoritmo usando essas sublistas que apenas para quando a lista passada como parâmetro for vazia. Na criação das sublistas usamos o método *sublist()* que tem uma complexidade de $\boldsymbol{O(1)}$.

Podemos então concluir que cada iteração do algoritmo recursivo tem uma complexidade de $\boldsymbol{O(nlog(n))}$ no pior caso, mas sendo um método recursivo que vai ter uma complexidade de $\boldsymbol{O(n)}$ (porque vai passar por todos os elementos da lista original) podemos concluir que este algoritmo total vai ter uma complexidade de $\boldsymbol{O(n^2\ log(n))}$ no pior caso.

US202



A User Story 202 pede que, dado um dado Callsign de um navio e uma dada data, seja obtido o porto mais próximo do dado navio na dada altura.

Inicialmente o algoritmo vai procurar na Shipstore, que é uma AVL, por um navio com o mesmo callsign, para isto é usado uma hashmap em que é guardado o MMSI de um dado navio juntamente com o seu callsign, ao procurar pelo callsign o hashmap retorna o respetivo MSSI, uma operação com complexidade O(1), e depois procura na arvore, que é ordenada pelo MMSI do navio, pelo navio escolhido. Como a árvore é balanceada esta procura tem uma complexidade de $\boldsymbol{O(log\ n)}$.

Posteriormente procura-se na positioningDataTree pela data mais próxima da desejada, como a árvore é balanceada e ordenada pela data este método tem uma complexidade $\boldsymbol{O(log\ n)}$.

Finalmente, tendo obtido a posição mais próxima da data desejada do navio, procura-se pelo porto mais próximo usando o método searchClosestStorage o que vai procurar o porto mais perto do dado ponto. Este método tem complexidade de pior case $\boldsymbol{O(n)}$ quando a KD-Tree tem dimensão de 4 ou esta mal-organizada. Porem esta arvore é balanceada e a dimensão é de 2 logo podemos assumir que a sua complexidade de médio é $\boldsymbol{O(log\ n)}$.

Desta forma a complexidade de pior case desta User Story é $\boldsymbol{O(n)}$ no pior caso e $\boldsymbol{O(log\ n)}$ no caso médio.